the mutually incompatible subjects. Since the solution obtained takes exactly fourteen periods it is the best possible in this case.

If a subject whose papers must not be in consecutive periods appears at the end of the table, this may lead to an excessive number of periods used. As suggested above this can be rectified by introducing some spurious subject numbers and making the offending subject incompatible with them. By this means it will be given a higher priority in the "grand sort" but will not affect the other conditions adversely.

The time taken for the above illustration on an 803 computer with an addition time of 576 μsec, was about $12\frac{1}{2}$ minutes. If all 340 subjects are used the program will take several hours, depending in a complex way upon the number of conditions imposed. If no conditions of types (ii) or (iii) of Section 2 are imposed the times are considerably reduced. For example, with 340 subjects with an average of $1\frac{1}{2}$ papers per subject, and with no imposed conditions of the type "paper *A* must not precede paper *B*," the time taken was about ninety minutes.

References

APPLEBY, J. S., BLAKE, D. V., and NEWMAN, E. A. (1961). "Techniques for producing School Timetables on a Computer and their Application to other Scheduling Problems," *The Computer Journal*, Vol. 3, p. 237.

CSIMA, J., and GOTLIEB, C. C. (1963). "A Computer Method for Constructing School Timetables." Presented at the Eighteenth Annual Conference of the Association for Computing Machinery, Denver, Colorado.

SHERMAN, G. R. (1963). "A Combinatorial Problem Arising from Scheduling of University Classes," *Journal of the Tennessee Academy of Science*, Vol. 38, No. 3, p. 115.

Note on a machine algorithm for conversion from reflected binary to natural binary

Frequently in special-purpose machines an input is provided in reflected binary (or Gray) code, but arithmetic in the processor is done in natural binary. Reflected binary arithmetic is too difficult to implement and cannot be done so economically at the present time. The problem is to convert from reflected to natural binary in the optimum fashion.

Call the reflected code register G and the natural binary register B, both of n bits. The bits of each are subscripted from least to greatest significance thus:

$$G = g_n g_{n-1} \ldots g_2 g_1 \tag{1}$$

$$B = b_n b_{n-1} \ldots b_2 b_1 \tag{2}$$

The conventional solution is to attach a logical converter between G and B. When actuated, this converter produces an output b_j representing the jth natural bit in terms of the jth, $(j+1)$th, ... nth reflected bit. This relation is succinctly expressed if we permit logic and arithmetic to be mixed,

$$b_j = \overline{\sum_{i=j+1}^{n} g_i \,(\text{mod } 2)}\, g_j \vee \sum_{i=j+1}^{n} g_i \,(\text{mod } 2) \bar{g}_j. \tag{3}$$

The summation is arithmetic and requires that all 1s between $j+1$ and n be added together. The expression "mod 2" requires that if this sum is odd, 1 is substituted for it; if even, 0 is substituted. The overbar means that 0 and 1 are interchanged beneath it.

Although the equation (3) has been simplified, the logic to do it has not. The logic to form a sum mod 2 is not complicated, but it is extensive, as those familiar with parity checkers will testify.

A simple relation exists between *this* natural bit, *this* reflected bit and the *previous* natural bit,

$$b_j = \overline{b_{j+1}} g_j \vee b_{j+1} \bar{g}_j. \tag{4}$$

Conversion is performed from left to right, more-significant bits being converted first. If the *last* bit g_{j+1} converted into a 0(b_{j+1} is 0), then *this* natural bit b_j is the same as *this* reflected bit g_j; if b_{j+1} was found to be 1, then g_j is reversed to get the proper value for b_j.

With (4), a simple command can be incorporated into the processor for the conversion. A counter designated i keeps track of the number of bits so far converted. This algorithm, particularly suitable to serial machines, is micro-programmed below:

(5) $0 \rightarrow B$

(6) $0 \rightarrow i$

(7) $\bar{b}_1 g_n \vee b_1 \bar{g}_n \rightarrow b_n$

(8) $B \overset{1}{\leftarrow} (B)$

(9) $G \overset{1}{\leftarrow} (G)$

(10) $i + 1 \rightarrow i$

(11) $i : n < \Rightarrow (7); = \Rightarrow$ end

To begin, the natural binary register is cleared (5) and the counter set to 0 (6). The left-hand (most-significant) bit of G and B is processed. The natural bit, b_n, is set according to (7). A circular shift of one place for G and B is done (8), (9), which moves each bit one position left; the *most*-significant bit, b_n or g_n, is moved to the *least* significant position, b_1 or g_1. The counter is incremented (10) and tested (11) to see if all bits have been converted. If not, the next bit is processed by entering the microprogram at (7).

Since B is cleared in (5), b_1 is 0 the first time (7) is performed, so that b_n is set to g_n. Other times (7) is done, b_n is set to g_n if the last natural bit (b_1) was 0 and to $\bar{g}_n$ otherwise.

IVAN FLORES.

Computer Design Consultant,
Norwalk, Connecticut, U.S.A.